

Fast Compatibility Testing for Phylogenies with Nested Taxa^{*}

Yun Deng[†]

David Fernández-Baca[‡]

Abstract

Semi-labeled trees are phylogenies whose internal nodes may be labeled by higher-order taxa. Thus, a leaf labeled *Mus musculus* could nest within a subtree whose root node is labeled Rodentia, which itself could nest within a subtree whose root is labeled Mammalia. Suppose we are given collection \mathcal{P} of semi-labeled trees over various subsets of a set of taxa. The ancestral compatibility problem asks whether there is a semi-labeled tree \mathcal{T} that respects the clusterings and the ancestor/descendant relationships implied by the trees in \mathcal{P} . We give a $\tilde{O}(M_{\mathcal{P}})$ algorithm for the ancestral compatibility problem, where $M_{\mathcal{P}}$ is the total number of nodes and edges in the trees in \mathcal{P} . Unlike the best previous algorithm, the running time of our method does not depend on the degrees of the nodes in the input trees.

1 Introduction

In the *tree compatibility problem*, we are given a collection $\mathcal{P} = \{\mathcal{T}_1, \dots, \mathcal{T}_k\}$ of rooted phylogenetic trees with partially overlapping taxon sets. \mathcal{P} is called a *profile* and the trees in \mathcal{P} are the *input trees*. The question is whether there exists a tree \mathcal{T} whose taxon set is the union of the taxon sets of the input trees, such that \mathcal{T} exhibits the clusterings implied by the input trees. That is, if two taxa are together in a subtree of some input tree, then they must also be together in some subtree of \mathcal{T} . The tree compatibility problem has been studied for over three decades [1, 8, 10, 20].

In the original version of the tree compatibility problem, only the leaves of the input trees are labeled. Here we study a generalization, called *ancestral compatibility*, in which taxa may be *nested*. That is, the internal nodes may also be labeled; these labels represent *higher-order taxa*, which are, in effect, sets of taxa. Thus, for example, an input tree may contain the taxon *Glycine max* (soybean) nested within a subtree whose root is labeled Fabaceae (the legumes), itself nested within an Angiosperm subtree. Note that leaves themselves may be labeled by higher-order taxa. The question now is whether there is a tree \mathcal{T} whose taxon set is the union of the taxon sets of the input trees, such that \mathcal{T} exhibits not only the clusterings among the taxa, but also the ancestor/descendant relationships among taxa in the input trees. Our main result is a $\tilde{O}(M_{\mathcal{P}})$ algorithm for the compatibility problem for trees with nested taxa, where $M_{\mathcal{P}}$ is the total number of nodes and edges in the trees in \mathcal{P} .

Background. The tree compatibility problem is a basic special case of the *supertree problem*. A supertree method is a way to synthesize a collection of phylogenetic trees with partially overlapping taxon sets into a single supertree that represents the information in the input trees. The supertree approach, proposed in the early 90s [2, 15], has been used successfully to build large-scale phylogenies [4].

^{*}Supported in part by the National Science Foundation under grant CCF-1422134.

[†]Department of Computer Science, Iowa State University, Ames, IA 50011, USA, yundeng@iastate.edu

[‡]Department of Computer Science, Iowa State University, Ames, IA 50011, USA, fernande@iastate.edu.

The original supertree methods were limited to input trees where only the leaves are labeled. Page [13] was among the first to note the need to handle phylogenies where internal nodes are labeled, and taxa are nested. A major motivation is the desire to incorporate *taxonomies* as input trees in large-scale supertree analyses, as way to circumvent one of the obstacles to building comprehensive phylogenies: the limited taxonomic overlap among different phylogenetic studies [16]. Taxonomies group organisms according to a system of taxonomic rank (e.g., family, genus, and species); two examples are the NCBI taxonomy [17] and the Angiosperm taxonomy [21]. Taxonomies spanning a broad range of taxa provide structure and completeness that might be hard to obtain otherwise. A recent example of the utility of taxonomies is the Open Tree of Life, a draft phylogeny for over 2.3 million species [11].

Taxonomies are not, strictly speaking, phylogenies. In particular, their internal nodes and some of their leaves are labeled with higher-order taxa. Nevertheless, taxonomies have many of the same mathematical characteristics as phylogenies. Indeed, both phylogenies and taxonomies are *semi-labeled trees* [5, 18]. We will use this term throughout the rest of the paper to refer to trees with nested taxa.

The fastest previous algorithm for testing ancestral compatibility, based on earlier work by Daniel and Semple [7], is due to Berry and Semple [3]. Their algorithm runs in $O(\log^2 n \cdot \tau_{\mathcal{P}})$ time using $O(\tau_{\mathcal{P}})$ space. Here, n is the number of distinct taxa in \mathcal{P} and $\tau_{\mathcal{P}} = \sum_{i=1}^k \sum_{v \in I(\mathcal{T}_i)} d(v)^2$, where $I(\mathcal{T}_i)$ is the set of internal nodes of \mathcal{T}_i , for each $i \in \{1, \dots, k\}$, and $d(v)$ is the degree of node v . While the algorithm is polynomial, its dependence on node degrees is problematic: semi-labeled trees can be highly unresolved (i.e., contain nodes of high degree), especially if they are taxonomies.

Our contributions. The $\tilde{O}(M_{\mathcal{P}})$ running time of our ancestral compatibility algorithm is independent of the degrees of the nodes of the input trees, a valuable characteristic for large datasets that include taxonomies. To achieve this time bound, we extend ideas from our recent algorithm for testing the compatibility of ordinary phylogenetic trees [8]. As in that algorithm, a central notion in the current paper is the *display graph* of profile \mathcal{P} , denoted $H_{\mathcal{P}}$. This is the graph obtained from the disjoint union of the trees in \mathcal{P} by identifying nodes that have the same label (see Section 4). The term “display graph” was introduced by Bryant and Lagergren [6], but similar ideas have been used elsewhere. In particular, the display graph is closely related to Berry and Semple’s *restricted descendanty graph* [3], a mixed graph whose directed edges correspond to the (undirected) edges of $H_{\mathcal{P}}$ and whose undirected edges have no correspondence in $H_{\mathcal{P}}$. The second kind of edges are the major component of the $\tau_{\mathcal{P}}$ term in the time and space complexity of Berry and Semple’s algorithm. The absence of such edges makes $H_{\mathcal{P}}$ significantly smaller than the restricted descendanty graph. Display graphs also bear some relation to *tree alignment graphs* [19].

Here, we exploit the display graph more extensively and more directly than our previous work. Although the display graph of a collection of semi-labeled trees is more complex than that of a collection of ordinary phylogenies, we are able to extend several of the key ideas — notably, that of a semi-universal label — to the general setting of semi-labeled trees. As in [8], the implementation relies on a dynamic graph data structure, but it requires a more careful amortized analysis based on a weighing scheme.

Contents. Section 2 presents basic definitions regarding semi-labeled trees and ancestral compatibility. . Section 3 introduces the display graph and discusses its properties. Section 4 presents BuildNT, our algorithm for testing ancestral compatibility. Section 5 gives the implementation details for BuildNT. Section 6 gives some concluding remarks.

2 Preliminaries

For each positive integer r , $[r]$ denotes the set $\{1, \dots, r\}$.

Let G be a graph. $V(G)$ and $E(G)$ denote the node and edge sets of G . The *degree* of a node $v \in V(G)$ is the number of edges incident on v . A *tree* is an acyclic connected graph. In this paper, all trees are assumed to be rooted. For a tree T , $r(T)$ denotes the root of T . Suppose $u, v \in V(T)$. Then, u is an *ancestor* of v in T , denoted $u \leq_T v$, if u lies on the path from v to $r(T)$ in T . If $u \leq_T v$, then v is a *descendant* of u . Node u is a *proper descendant* of v if u is a descendant of v and $v \neq u$. If $\{u, v\} \in E(T)$ and $u \leq_T v$, then u is the *parent* of v and v is a *child* of u . If neither $u \leq_T v$ nor $v \leq_T u$ hold, then we write $u \parallel_T v$ and say that u and v are *not comparable* in T .

Semi-labeled trees. A *semi-labeled tree* is a pair $\mathcal{T} = (T, \phi)$ where T is a tree and ϕ is a mapping from a set $L(\mathcal{T})$ to $V(T)$ such that, for every node $v \in V(T)$ of degree at most two, $v \in \phi(L(\mathcal{T}))$. $L(\mathcal{T})$ is the *label set* of \mathcal{T} and ϕ is the *labeling function* of \mathcal{T} .

For every node $v \in V(T)$, $\phi^{-1}(v)$ denotes the (possibly empty) subset of $L(\mathcal{T})$ whose elements map into v ; these elements are the *labels* of v (thus, each label is a taxon). If $\phi^{-1}(v) \neq \emptyset$, then v is *labeled*; otherwise, v is *unlabeled*. Note that, by definition, every leaf in a semi-labeled tree is labeled. Further, any node, including the root, that has a single child must be labeled. Nodes with two or more children may be labeled or unlabeled. A semi-labeled tree $\mathcal{T} = (T, \phi)$ is *singularly labeled* if every node in T has at most one label; \mathcal{T} is *fully labeled* if every node in T is labeled.

Semi-labeled trees, also known as *X-trees*, generalize ordinary phylogenetic trees, also known as *phylogenetic X-trees* [18]. An ordinary phylogenetic tree is a semi-labeled tree $\mathcal{T} = (T, \phi)$ where $r(T)$ has degree at least two and ϕ is a bijection from $L(\mathcal{T})$ into leaf set of T (thus, internal nodes are not labeled).

Let $\mathcal{T} = (T, \phi)$ be a semi-labeled tree and let ℓ and ℓ' be two labels in $L(\mathcal{T})$. If $\phi(\ell) \leq_T \phi(\ell')$, then we write $\ell \leq_{\mathcal{T}} \ell'$, and say that ℓ' is a *descendant* of ℓ in \mathcal{T} and that ℓ is an *ancestor* of ℓ' . We write $\ell <_{\mathcal{T}} \ell'$ if $\phi(\ell')$ is a proper descendant of $\phi(\ell)$. If $\phi(\ell) \parallel_T \phi(\ell')$, then we write $\ell \parallel_{\mathcal{T}} \ell'$ and say that ℓ and ℓ' are *not comparable* in \mathcal{T} . If \mathcal{T} is fully labeled and $\phi(\ell)$ is the parent of $\phi(\ell')$ in T , then ℓ is the *parent* of ℓ' in \mathcal{T} and ℓ' is a *child* of ℓ in \mathcal{T} ; two labels with the same parent are *siblings*.

Two semi-labelled trees $\mathcal{T} = (T, \phi)$ and $\mathcal{T}' = (T', \phi')$ are *isomorphic* if there exists a bijection $\psi : V(T) \rightarrow V(T')$ such that $\phi' = \psi \circ \phi$ and, for any two nodes $u, v \in V(T)$, $(u, v) \in E(T)$ if and only if $(\psi(u), \psi(v)) \in E(T')$.

Let $\mathcal{T} = (T, \phi)$ be a semi-labeled tree. For each $u \in V(T)$, $X(u)$ denotes the set of all labels in the subtree of T rooted at u ; that is, $X(u) = \bigcup_{v: u \leq_T v} \phi^{-1}(v)$. $X(u)$ is called a *cluster* of T . $\text{Cl}(\mathcal{T})$ denotes the set of all clusters of \mathcal{T} . It is well known [18, Theorem 3.5.2] that a semi-labeled tree \mathcal{T} is completely determined by $\text{Cl}(\mathcal{T})$. That is, if $\text{Cl}(\mathcal{T}) = \text{Cl}(\mathcal{T}')$ for some other semi-labeled tree \mathcal{T}' , then \mathcal{T} is isomorphic to \mathcal{T}' .

Suppose $A \subseteq L(\mathcal{T})$ for a semi-labeled tree $\mathcal{T} = (T, \phi)$. The *restriction* of \mathcal{T} to A , denoted $\mathcal{T}|A$, is the semi-labeled tree whose cluster set is $\text{Cl}(\mathcal{T}|A) = \{X \cap A : X \in \text{Cl}(\mathcal{T}) \text{ and } X \cap A \neq \emptyset\}$. Intuitively, $\mathcal{T}|A$ is obtained from the minimal rooted subtree of T that connects the nodes in $\phi(A)$ by suppressing all vertices of degree two that are not in $\phi(A)$.

Let $\mathcal{T} = (T, \phi)$ and $\mathcal{T}' = (T', \phi')$ be semi-labeled trees such that $L(\mathcal{T}') \subseteq L(\mathcal{T})$. \mathcal{T} *ancestrally displays* \mathcal{T}' if $\text{Cl}(\mathcal{T}') \subseteq \text{Cl}(\mathcal{T}|L(\mathcal{T}'))$. Equivalently, \mathcal{T} *ancestrally displays* \mathcal{T}' if \mathcal{T}' can be obtained from $\mathcal{T}|L(\mathcal{T}')$ by contracting edges, and, for any $\ell_1, \ell_2 \in L(\mathcal{T}')$, (i) if $\ell_1 <_{\mathcal{T}'} \ell_2$, then $\ell_1 <_{\mathcal{T}} \ell_2$, and (ii) if $\ell_1 \parallel_{\mathcal{T}'} \ell_2$, then $\ell_1 \parallel_{\mathcal{T}} \ell_2$. The notion of “ancestrally displays” for semi-labeled trees generalizes the well-known notion of “displays” for ordinary phylogenetic trees [18].

For a semi-labelled tree \mathcal{T} , let

$$D(\mathcal{T}) = \{(\ell, \ell') : \ell, \ell' \in L(\mathcal{T}) \text{ and } \ell <_{\mathcal{T}} \ell'\} \quad \text{and} \quad N(\mathcal{T}) = \{\{\ell, \ell'\} : \ell, \ell' \in L(\mathcal{T}) \text{ and } \ell \parallel_{\mathcal{T}} \ell'\}. \quad (1)$$

Note that $D(\mathcal{T})$ consists of *ordered* pairs, while $N(\mathcal{T})$ consists of *unordered* pairs.

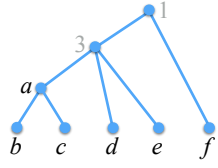


Figure 1: A profile $\mathcal{P} = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$ — trees are ordered left-to-right. The letters are the original labels; grey numbers are labels added to make the trees fully labeled. (Adapted from [3].)

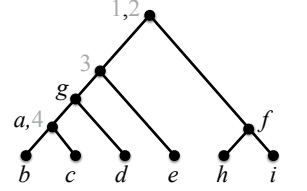


Figure 2: A tree \mathcal{T} that ancestrally displays the profile of Figure 1. (Adapted from [3].)

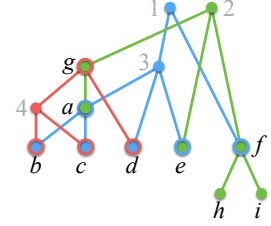


Figure 3: The display graph $H_{\mathcal{P}}$ for the profile of Figure 1.

Lemma 1 (Bordewich et al. [5]). *Let \mathcal{T} and \mathcal{T}' be semi-labelled trees such that $L(\mathcal{T}') \subseteq L(\mathcal{T})$. Then \mathcal{T} ancestrally displays \mathcal{T}' if and only if $D(\mathcal{T}') \subseteq D(\mathcal{T})$ and $N(\mathcal{T}') \subseteq N(\mathcal{T})$.*

Profiles and ancestral compatibility. Throughout the rest of this paper $\mathcal{P} = \{\mathcal{T}_1, \dots, \mathcal{T}_k\}$ denotes a set where, for each $i \in [k]$, $\mathcal{T}_i = (T_i, \phi_i)$ is a semi-labelled tree. We refer to \mathcal{P} as a *profile*, and write $L(\mathcal{P})$ to denote $\bigcup_{i \in [k]} L(\mathcal{T}_i)$, the *label set* of \mathcal{P} . Figure 1 shows a profile where $L(\mathcal{P}) = \{a, b, c, d, e, f, g, h, i\}$. We write $V(\mathcal{P})$ for $\bigcup_{i \in [k]} V(T_i)$ and $E(\mathcal{P})$ for $\bigcup_{i \in [k]} E(T_i)$. The *size* of \mathcal{P} is $M_{\mathcal{P}} = |V(\mathcal{P})| + |E(\mathcal{P})|$.

\mathcal{P} is *ancestrally compatible* if there is a rooted semi-labelled tree \mathcal{T} that ancestrally displays each of the trees in \mathcal{P} . If \mathcal{T} exists, we say that \mathcal{T} *ancestrally displays* \mathcal{P} (see Figure 2).

Given a subset X of $L(\mathcal{P})$, the *restriction* of \mathcal{P} to X , denoted $\mathcal{P}|X$, is the profile $\{\mathcal{T}_1|X \cap L(\mathcal{T}_1), \dots, \mathcal{T}_k|X \cap L(\mathcal{T}_k)\}$. The proof of the following lemma is straightforward.

Lemma 2. *Suppose \mathcal{P} is ancestrally compatible and let \mathcal{T} be a tree that ancestrally displays \mathcal{P} . Then, for any $X \subseteq L(\mathcal{P})$, $\mathcal{T}|X$ ancestrally displays $\mathcal{P}|X$.*

A semi-labeled tree $\mathcal{T} = (T, \phi)$ is *fully labeled* if every node in T is labeled. Suppose \mathcal{P} contains trees that are not fully labeled. We can convert \mathcal{P} into an equivalent profile \mathcal{P}' of fully-labeled trees as follows. For each $i \in [k]$, let l_i be the number of unlabeled nodes in T_i . Create a set L' of $n' = \sum_{i \in [k]} l_i$ labels such that $L' \cap L(\mathcal{P}) = \emptyset$. For each $i \in [k]$ and each $v \in V(T_i)$ such that $\phi_i^{-1}(v) = \emptyset$, make $\phi_i^{-1}(v) = \{\ell\}$, where ℓ is a distinct element from L' . We refer to \mathcal{P}' as the *profile obtained by adding distinct new labels to \mathcal{P}* (see Figure 1).

Lemma 3 (Daniel and Semple [7]). *Let \mathcal{P}' be the profile obtained by adding distinct new labels to \mathcal{P} . Then, \mathcal{P} is ancestrally compatible if and only if \mathcal{P}' is ancestrally compatible. Further, if \mathcal{T} is a semi-labeled phylogenetic tree that ancestrally displays \mathcal{P}' , then \mathcal{T} ancestrally displays \mathcal{P} .*

From this point forward, we shall assume that, for each $i \in [k]$, \mathcal{T}_i is fully and singularly labeled. By Lemma 3, no generality is lost in assuming that all trees in \mathcal{P} are fully labeled. The assumption that the trees are singularly labeled is inessential; it is only for clarity. Note that, even with the latter assumption, a tree that ancestrally displays \mathcal{P} is not necessarily singularly labeled. Figure 2 illustrates this fact.

3 The Display Graph

The *display graph* of a profile \mathcal{P} , denoted $H_{\mathcal{P}}$, is the graph obtained from the disjoint union of the underlying trees T_1, \dots, T_k by identifying nodes that have the same label. Multiple edges between the same pair of nodes are replaced by a single edge. See Figure 3.

$H_{\mathcal{P}}$ has $O(M_{\mathcal{P}})$ nodes and edges, and can be constructed in $O(M_{\mathcal{P}})$ time. By our assumption that all the trees in \mathcal{P} are fully and singularly labeled, there is a bijection between the labels in $L(\mathcal{P})$ and the nodes of $H_{\mathcal{P}}$. Thus, from this point forward, we refer to the nodes of $H_{\mathcal{P}}$ by their labels. It is easy to see that if $H_{\mathcal{P}}$ is not connected, then \mathcal{P} decomposes into label-disjoint sub-profiles, and that \mathcal{P} is compatible if and only if each sub-profile is compatible. Thus, we shall assume, without loss of generality, that $H_{\mathcal{P}}$ is connected.

Positions. A *position* (for \mathcal{P}) is a vector $U = (U(1), \dots, U(k))$, where $U(i) \subseteq L(\mathcal{T}_i)$, for each $i \in [k]$. Since labels may be shared among trees, we may have $U(i) \cap U(j) \neq \emptyset$, for $i, j \in [k]$ with $i \neq j$. For each $i \in [k]$, let $\text{Desc}_i(U) = \{\ell : \ell' \leq_{\mathcal{T}_i} \ell, \text{ for some } \ell' \in U(i)\}$, and let $\text{Desc}_{\mathcal{P}}(U) = \bigcup_{i \in [k]} \text{Desc}_i(U)$.

A position U is *valid* if, for each $i \in [k]$,

(V1) if $|U(i)| \geq 2$, then the elements of $U(i)$ are siblings in \mathcal{T}_i and

(V2) $\text{Desc}_i(U) = \text{Desc}_{\mathcal{P}}(U) \cap L(\mathcal{T}_i)$.

Lemma 4. For any valid position U , $\mathcal{P}|\text{Desc}_{\mathcal{P}}(U) = \{\mathcal{T}_1|\text{Desc}_1(U), \dots, \mathcal{T}_k|\text{Desc}_k(U)\}$.

Proof. By (V2), we have that $\mathcal{T}_i|\text{Desc}_i(U)$ and $\mathcal{T}_i|\text{Desc}_{\mathcal{P}}(U) \cap L(\mathcal{T}_i)$ are isomorphic, for each $i \in [k]$. The lemma then follows from the definition of $\mathcal{P}|\text{Desc}_{\mathcal{P}}(U)$. \square

For any valid position U , $H_{\mathcal{P}}(U)$ denotes the subgraph of $H_{\mathcal{P}}$ induced by $\text{Desc}_{\mathcal{P}}(U)$.

Observation 1. For any valid position U , $H_{\mathcal{P}}(U)$ is the subgraph of $H_{\mathcal{P}}$ obtained by deleting all labels in $V(H_{\mathcal{P}}) \setminus \text{Desc}_{\mathcal{P}}(U)$, along with all incident edges.

A valid position of special interest to us is U_{root} , where $U_{\text{root}}(i) = \phi_i^{-1}(r(\mathcal{T}_i))$, for each $i \in [k]$. That is, $U_{\text{root}}(i)$ is a singleton containing only the label of $r(\mathcal{T}_i)$. Thus, in Figure 3, $(U_{\text{root}}(1), U_{\text{root}}(2), U_{\text{root}}(3)) = (\{1\}, \{2\}, \{g\})$. It is straightforward to verify that U_{root} is indeed valid, that $\text{Desc}_{\mathcal{P}}(U_{\text{root}}) = L(\mathcal{P})$, and that $H_{\mathcal{P}}(U_{\text{root}}) = H_{\mathcal{P}}$.

Semi-universal labels. Let U be a valid position, and let ℓ be a label in U . Then, ℓ is *semi-universal* in U if $U(i) = \{\ell\}$, for every $i \in [k]$ such that $\ell \in L(\mathcal{T}_i)$. It can be verified that in Figure 3, labels 1 and 2 are semi-universal in U_{root} , but g is not, since g is in both $L(\mathcal{T}_2)$ and $L(\mathcal{T}_3)$, but $U_{\text{root}}(2) \neq \{g\}$.

The term “semi-universal”, borrowed from Pe’er et al. [14], derives from the following fact. Suppose that \mathcal{P} is ancestrally compatible, that \mathcal{T} is a tree that ancestrally displays \mathcal{P} , and that ℓ is a semi-universal label for some valid position U . Then, as we shall see, ℓ must label the root u_{ℓ} of a subtree of \mathcal{T} that contains all the descendants of ℓ in \mathcal{T}_i , for every i such that $\ell \in L(\mathcal{T}_i)$. The qualifier “semi” is because this subtree may also contain labels that do not descend from ℓ in any input tree, but descend from some other semi-universal label ℓ' in U instead. In this case, ℓ' also labels u_{ℓ} . This property of semi-universal labels is exploited in both our ancestral compatibility algorithm and its proof of correctness (see Section 4).

For each label $\ell \in L(\mathcal{P})$, let k_{ℓ} denote the number of input trees that contain label ℓ . We can obtain k_{ℓ} for every $\ell \in L(\mathcal{P})$ in $O(M_{\mathcal{P}})$ time during the construction of $H_{\mathcal{P}}$.

Lemma 5. Let $U = (U(1), \dots, U(k))$ be a valid position. Then, label ℓ is semi-universal in U if the cardinality of the set $J_{\ell} = \{i \in [k] : U(i) = \{\ell\}\}$ equals k_{ℓ} .

Proof. By definition, $U(i) = \{\ell\}$, for every $i \in J_{\ell}$. Since $|J_{\ell}| = k_{\ell}$, the lemma follows. \square

Successor positions. For every $i \in [k]$ and every $\ell \in L(\mathcal{T}_i)$, let $\text{Ch}_i(\ell)$ denote the set of children of ℓ in $L(\mathcal{T}_i)$. For a subset A of $L(\mathcal{T}_i)$, let $\text{Ch}_i(A) = \bigcup_{\ell \in A} \text{Ch}_i(\ell)$. Let U be a valid position, and S be the set of semi-universal labels in U . The *successor of U with respect to S* is the position U' defined as follows. For each $\ell \in S$ and each $i \in [k]$, if $U(i) = \{\ell\}$, then $U'(i) = \text{Ch}_i(\ell)$; otherwise, $U'(i) = U(i)$.

In Figure 3, the set of semi-universal labels in U_{root} is $S = \{1, 2\}$. Since $\text{Ch}_1(1) = \{3, f\}$ and $\text{Ch}_2(2) = \{e, f, g\}$, the successor of U_{root} is $U' = (\{3, f\}, \{e, f, g\}, \{g\})$.

Observation 2. Let U be a valid position, and let U' be the successor of U with respect to the set S of semi-universal labels in U . Then, $H_{\mathcal{P}}(U')$ can be obtained from $H_{\mathcal{P}}(U)$ by doing the following for each $\ell \in S$: (1) for each $i \in [k]$ such that $U(i) = \{\ell\}$, delete all edges between ℓ and $\text{Ch}_i(\ell)$; (2) delete ℓ .

Let U be a valid position, and W be a subset of $\text{Desc}_{\mathcal{P}}(U)$. Then, $U|W$ denotes the position $(U(1) \cap W, \dots, U(k) \cap W)$. In Figure 3, the components of $H_{\mathcal{P}}(U')$, where U' is the successor of U_{root} , are $W_1 = \{3, 4, a, b, c, d, e, g\}$ and $W_2 = \{f, h, i\}$. Thus, $U'|W_1 = (\{3\}, \{e, g\}, \{g\})$ and $U'|W_2 = (\{f\}, \{f\}, \emptyset)$. We have the following result.

Lemma 6. Let U be a valid position, and S be the set of all semi-universal labels in U . Let U' be the successor of U with respect to S , and let W_1, W_2, \dots, W_p be the label sets of the connected components of $H_{\mathcal{P}}(U')$. Then, $U'|W_j$ is a valid position, for each $j \in [p]$.

Proof. It suffices to argue that U' satisfies conditions (V1) and (V2). The lemma then follows from the fact that the connected components of $H_{\mathcal{P}}(U')$ are label-disjoint.

U' must satisfy condition (V1), since U does. Suppose $\ell \in S$. Then, for each $i \in [k]$ such that $\ell \in L(\mathcal{T}_i)$, $\text{Desc}_i(U') = \text{Desc}_i(U) \setminus \{\ell\}$ and $\text{Desc}_{\mathcal{P}}(U') \cap L(\mathcal{T}_i) = (\text{Desc}_{\mathcal{P}}(U) \cap L(\mathcal{T}_i)) \setminus \{\ell\}$. Thus, since (V2) holds for U , it also holds for U' . \square

4 Testing Ancestral Compatibility

BuildNT (Algorithm 1) is our algorithm for testing compatibility of semi-labeled trees. Its argument, U , is a valid position in \mathcal{P} such that $H_{\mathcal{P}}(U)$ is connected. Line 1 computes the set S of semi-universal labels in U . If S is empty, then, as argued in Theorem 1 below, $\mathcal{P}|\text{Desc}_{\mathcal{P}}(U)$ is incompatible, and, thus, so is \mathcal{P} . This fact is reported in Line 3. Line 4 checks if S contains exactly one label ℓ , with no proper descendants. If so, by the connectivity assumption, ℓ must be the only element in $\text{Desc}_{\mathcal{P}}(U)$. Therefore, Line 5 simply returns the tree with a single node, labeled ℓ . Line 6 updates U , replacing it by its successor with respect to S . Let W_1, \dots, W_p be the connected components of $H_{\mathcal{P}}(U)$ after updating U . By Lemma 6, $U|W_j$ is a valid position, for each $j \in [p]$. Lines 7–11 recursively invoke BuildNT on $U|W_j$ for each $j \in [p]$, to determine if there is a tree t_j that ancestrally displays $\mathcal{P}|\text{Desc}_{\mathcal{P}}(U \cap W_j)$. If any subproblem is incompatible, Line 11 reports that \mathcal{P} is incompatible. Otherwise, Lines 12–13 assemble the t_j s into a single tree that displays $\mathcal{P}|\text{Desc}_{\mathcal{P}}(U)$, whose root is labeled by the semi-universal labels in the set S of Line 1.

Next, we argue the correctness of BuildNT.

Theorem 1. Let $\mathcal{P} = \{\mathcal{T}_1, \dots, \mathcal{T}_k\}$ be a profile and let $U_{\text{root}} = (U_{\text{root}}(1), \dots, U_{\text{root}}(k))$, where, for each $i \in [k]$, $U_{\text{root}}(i) = \phi_i^{-1}(r(\mathcal{T}_i))$. Then, BuildNT(U_{root}) returns either (i) a semi-labeled tree \mathcal{T} that ancestrally displays \mathcal{P} , if \mathcal{P} is ancestrally compatible, or (ii) incompatible otherwise.

Proof. (i) Suppose that BuildNT(U_{root}) outputs a semi-labeled tree \mathcal{T} . We prove that \mathcal{T} ancestrally displays \mathcal{P} . By Lemma 1, it suffices to show that $D(\mathcal{T}_i) \subseteq D(\mathcal{T})$ and $N(\mathcal{T}_i) \subseteq N(\mathcal{T})$, for each $i \in [k]$.

Algorithm 1: BuildNT(U)

Input: A valid position U for \mathcal{P} such that $H_{\mathcal{P}}(U)$ is connected.

Output: A semi-labeled tree that ancestrally displays $\mathcal{P}' = \mathcal{P}|_{\text{Desc}_{\mathcal{P}}(U)}$, if \mathcal{P}' is ancestrally compatible; incompatible otherwise.

```
1 Let  $S = \{\ell \in U : \ell \text{ is semi-universal in } U\}$ 
2 if  $S = \emptyset$  then
3   | return incompatible
4 if  $|S| = 1$  and the single element, } \ell, \text{ of } S \text{ has no proper descendants} then
5   | return the tree consisting of exactly one node, whose label set is  $\{\ell\}$ 
6 Replace  $U$  by the successor of  $U$  with respect to  $S$ .
7 Let  $W_1, W_2, \dots, W_p$  be the connected components of  $H_{\mathcal{P}}(U)$ 
8 foreach  $j \in [p]$  do
9   | Let  $t_j = \text{BuildNT}(U|W_j)$ 
10  | if  $t_j$  is not a tree then
11    | return incompatible
12 Create a node  $r_U$ , whose label set is  $S$ 
13 return the tree with root  $r_U$  and subtrees  $t_1, \dots, t_p$ 
```

Consider any $(\ell, \ell') \in D(\mathcal{T}_i)$. Then, ℓ has a child ℓ'' in \mathcal{T}_i such that $\ell'' \leq_{\mathcal{T}_i} \ell'$. There must be a recursive call to BuildNT(U), for some valid position U , where ℓ is the set S of semi-universal labels obtained in Line 1. By Observation 2, label ℓ'' , and thus ℓ' , both lie in one of the connected components of the graph obtained by deleting all labels in S , including ℓ , and their incident edges from $H_{\mathcal{P}}(U)$. It now follows from the construction of \mathcal{T} that $(\ell, \ell') \in D(\mathcal{T})$. Thus, $D(\mathcal{T}_i) \subseteq D(\mathcal{T})$.

Now, consider any $\{\ell, \ell'\} \in N(\mathcal{T}_i)$. Let v be the lowest common ancestor of $\phi_i(\ell)$ and $\phi_i(\ell')$ in \mathcal{T}_i and let ℓ_v be the label of v . Then, ℓ_v has a pair of children, ℓ_1 and ℓ_2 say, in \mathcal{T}_i such that $\ell_1 \leq_{\mathcal{T}_i} \ell$, and $\ell_2 \leq_{\mathcal{T}_i} \ell'$. Because BuildNT(U_{root}) returns a tree, there are recursive calls BuildNT(U_1) and BuildNT(U_2) for valid positions U_1 and U_2 such that ℓ_1 is semi-universal for U_1 and ℓ_2 is semi-universal for U_2 . We must have $U_1 \neq U_2$; otherwise, $|U_1(i)| = |U_2(i)| \geq 2$, and, thus, neither ℓ_1 nor ℓ_2 is semi-universal, a contradiction. Further, it follows from the construction of \mathcal{T} that we must have $\text{Desc}_{\mathcal{P}}(U_1) \cap \text{Desc}_{\mathcal{P}}(U_2) = \emptyset$. Hence, $\ell \parallel_{\mathcal{T}} \ell'$, and, therefore, $\{\ell, \ell'\} \in N(\mathcal{T})$.

(ii) Assume, by way of contradiction, that BuildNT(U_{root}) returns incompatible, but that \mathcal{P} is ancestrally compatible. By assumption, there exists a semi-labeled tree \mathcal{T} that ancestrally displays \mathcal{P} . Since BuildNT(U_{root}) returns incompatible, there is a recursive call to BuildNT(U) for some valid position U such that U has no semi-universal label, and the set S of Line 1 is empty.

By Lemma 2, $\mathcal{T}|_{\text{Desc}_{\mathcal{P}}(U)}$ ancestrally displays $\mathcal{P}|_{\text{Desc}_{\mathcal{P}}(U)}$. Thus, by Lemma 4, $\mathcal{T}|_{\text{Desc}_{\mathcal{P}}(U)}$ ancestrally displays $\mathcal{T}_i|_{\text{Desc}_i(U)}$, for every $i \in [k]$. Let ℓ be any label in the label set of the root of $\mathcal{T}|_{\text{Desc}_{\mathcal{P}}(U)}$. Then, for each $i \in [k]$ such that $\ell \in L(\mathcal{T}_i)$, ℓ must be the label of the root of $\mathcal{T}_i|_{\text{Desc}_i(U)}$. Thus, for each such i , $U(i) = \{\ell\}$. Hence, ℓ is semi-universal in U , a contradiction. \square

5 Implementation

Here we describe an efficient implementation of BuildNT. We focus on two key aspects: finding semi-universal labels in Line 1, and updating U and $H_{\mathcal{P}}(U)$ in Lines 6 and 7.

By Observation 1, at each recursive call, BuildNT deals with a graph obtained from $H_{\mathcal{P}}$ through edge and node deletions. To handle these deletions efficiently, we represent $H_{\mathcal{P}}$ using the dynamic graph connectivity data structure of Holm et al. [12], which we refer to as *HDT*. HDT allows us to maintain the list of nodes in each component, as well as the number of these nodes so that, if we start with no edges in a graph with N nodes, the amortized cost of each update is $O(\log^2 N)$. Since $H_{\mathcal{P}}$ has $O(M_{\mathcal{P}})$ nodes, each update takes $O(\log^2 M_{\mathcal{P}})$ time. The total number of edge and node deletions performed by BuildNT(U_{root}) — including all deletions in the recursive calls — is at most the total number of edges and nodes in $H_{\mathcal{P}}$, which is $O(M_{\mathcal{P}})$. HDT allows us to maintain connectivity information throughout the entire algorithm in $O(M_{\mathcal{P}} \log^2 M_{\mathcal{P}})$ time.

As deletions are performed on $H_{\mathcal{P}}$, BuildNT maintains three data fields for each connected component Y that is created: $Y.\text{weight}$, $Y.\text{map}$, and $Y.\text{semiU}$. It also maintains a field $\ell.\text{count}$, for each $\ell \in L(\mathcal{P})$.

1. $Y.\text{weight}$ equals $\sum_{\ell \in Y} k_{\ell}$.
2. $Y.\text{map}$ is a map from a set $J_Y \subseteq [k]$ to a set of nonempty subsets of $Y \cap L(T_i)$. For each $i \in J_Y$, $Y.\text{map}(i)$ denotes the set associated with i .
3. $\ell.\text{count}$ equals the cardinality of the set $\{i \in [k] : Y.\text{map}(i) \text{ is defined and } Y.\text{map}(i) = \{\ell\}\}$. (Recall that k_{ℓ} is the number of input trees that contain ℓ .)
4. $Y.\text{semiU}$ is a set containing all labels $\ell \in Y$ such that $\ell.\text{count} = k_{\ell}$.

Informally, each set $Y.\text{map}(i)$ corresponds to a non-empty $U(i)$; $Y.\text{semiU}$ corresponds to the semi-universal labels in Y . Next, we formalize these ideas.

At the start of the execution of BuildNT(U) for any valid position U , $H_{\mathcal{P}}(U)$ has a single connected component, $Y_U = \text{Desc}_{\mathcal{P}}(U)$. Our implementation maintains the following invariant.

INV: At the beginning of the execution of BuildNT(U), $Y_U.\text{map}(i) = U(i)$ for each $i \in [k]$ such that $U(i) \neq \emptyset$, and $Y_U.\text{map}(i)$ is undefined for each $i \in [k]$ such that $U(i) = \emptyset$.

Thus, $\ell.\text{count}$ equals the number of indices $i \in [k]$ such that $U(i) = \{\ell\}$. Along with Lemma 5, INV implies that, at the beginning of the execution of BuildNT(U), $Y_U.\text{semiU}$ contains precisely the semi-universal labels of U . Thus, the set S of line 1 of BuildNT(U) can be retrieved in $O(1)$ time.

To establish INV for the initial valid position U_{root} , we proceed as follows. By assumption, $H_{\mathcal{P}}(U_{\text{root}})$ has a single connected component, $Y_{\text{root}} = L(\mathcal{P})$. Since $H_{\mathcal{P}}(U_{\text{root}})$ equals $H_{\mathcal{P}}$, we initialize data fields 1–4 for Y_{root} during the construction of $H_{\mathcal{P}}$. $Y_{\text{root}}.\text{weight}$ is simply $\sum_{\ell \in L(\mathcal{P})} k_{\ell}$. For each $i \in [k]$, $Y_{\text{root}}.\text{map}(i)$ is $\{\ell\}$, where ℓ is the label of the root of T_i . We initialize the count fields as follows. First, set $\ell.\text{count}$ to 0 for all $\ell \in L(\mathcal{P})$. Then, iterate through each $i \in [k]$, incrementing $\ell.\text{count}$ by one if $Y_{\text{root}}.\text{map}(i) = \{\ell\}$. Finally, $Y_{\text{root}}.\text{semiU}$ consists of all $\ell \in U_{\text{root}}$ such that $\ell.\text{count} = k_{\ell}$. All data fields can be initialized in $O(M_{\mathcal{P}})$ time.

We now focus on Lines 6 and 7 of BuildNT. By Observation 2, we can update U and $H_{\mathcal{P}}(U)$ jointly as follows. We use a temporary variable G_{BNT} . Prior to executing Line 6, we set $G_{\text{BNT}} = H_{\mathcal{P}}(U)$. Then, we successively consider each label $\ell \in S$, and perform two steps: (i) initialize data fields 1–4 in preparation for the deletion of ℓ and (ii) delete from G_{BNT} the edges incident on ℓ and then ℓ itself, updating data fields 1–4 as necessary, to maintain INV. After these steps are executed, G_{BNT} will equal $H_{\mathcal{P}}(U)$ for the new set U created by Line 6. Steps (i) and (ii) are done by Initialize(ℓ) (Algorithm 2) and Delete(ℓ) (Algorithm 3), respectively.

Lines 1–5 of Initialize(ℓ) initialize $Y.\text{map}$ and $Y.\text{semiU}$ to reflect the fact that label $\ell \in S$ is leaving $U(i)$, for each $i \in [k]$ such that $i \in L(T_i)$, to be replaced by its children in T_i , and will no longer be semi-universal. Lines 6–9 are needed to update certain count fields due to the possibility that singleton sets $Y.\text{map}(i)$ may be created in the preceding steps. The number of operations on $Y.\text{map}$ performed by

$\text{Initialize}(\ell)$ is $O(\sum_{i \in [k]: \ell \in L(\mathcal{T}_i)} |\text{Ch}_i(\ell)|)$; i.e., it is proportional to the total number of children of ℓ in all the input trees. Since ℓ is considered only once, the total number of operations on map fields of the various sets Y considered over the entire execution of $\text{BuildNT}(U_{\text{root}})$ is $O(M_{\mathcal{P}})$. The number of updates of $Y.\text{map}$ done by $\text{Initialize}(\ell)$ is at most k_ℓ ; the total over all ℓ considered by $\text{BuildNT}(U_{\text{root}})$ over its entire execution is $O(M_{\mathcal{P}})$.

Algorithm 2: $\text{Initialize}(\ell)$

```

1 Delete  $\ell$  from  $Y.\text{semiU}$ 
2 foreach  $i \in [k]$  such that  $\ell \in L(\mathcal{T}_i)$  do
3   Delete  $Y.\text{map}(i)$ 
4   foreach  $\alpha \in \text{Ch}_i(\ell)$  do
5     Add  $\alpha$  to  $Y.\text{map}(i)$ 
6   if  $Y.\text{map}(i)$  is a singleton then
7     Let  $\beta$  be the single element in  $Y.\text{map}(i)$ 
8     Set  $\beta.\text{count} = \beta.\text{count} + 1$ 
9     if  $\beta.\text{count} = k_\beta$  then add  $\beta$  to  $Y.\text{semiU}$ 

```

$\text{Delete}(\ell)$ begins by consulting HDT to identify the connected component Y that currently contains ℓ . The loop in Lines 2–19 successively deletes each edge between ℓ and a child α of ℓ , updating the appropriate data fields for the resulting connected components. Line 4 queries the HDT data structure to determine whether deleting (ℓ, α) splits Y into two components. If Y remains connected, no updates are needed — the **continue** statement skips the rest of the current iteration and proceeds directly to the next. Otherwise, Y is split into two parts Y_1 and Y_2 . Delete uses a weighted version of the technique of scanning the smaller component [9]. Line 4 identifies which of the two components has the smaller *weight* field; without loss of generality, it assumes that $Y_1.\text{weight} \leq Y_2.\text{weight}$. Lines 5 and 6 initialize $Y_1.\text{map}$ and $Y_1.\text{semiU}$ to null and $Y_2.\text{map}$ and $Y_2.\text{semiU}$ to the corresponding fields of Y . Lines 8–11, scan each label β in Y_1 , from $Y_2.\text{map}(i)$ to $Y_1.\text{map}(i)$, for every i such that $\beta \in L(\mathcal{T}_i)$. Set J , updated in Line 11, keeps track of the indices i such that $Y_1.\text{map}(i)$ and $Y_2.\text{map}(i)$ are modified. Lines 12–19 iterate through J to determine if any new singleton sets were created in either Y_1 or Y_2 . This operation requires at most one update in each of $Y_1.\text{semiU}$ and $Y_2.\text{semiU}$; each update takes $O(1)$ time. After all edges incident on ℓ are deleted, ℓ itself is deleted (Line 20).

The preceding description of $\text{Delete}(\ell)$ omits the updating of the *weight* fields of the connected components created by an edge deletion. This is done before Line 4, by (again) using the technique of scanning the smaller component. We consult HDT to determine which of Y_1 and Y_2 has fewer labels. Assuming, without loss of generality, that $|Y_1| < |Y_2|$, compute $Y_1.\text{weight}$ in a sequential scan of Y_1 . Then, $Y_2.\text{weight} = Y.\text{weight} - Y_1.\text{weight}$.

Let us track the number of operations on map fields in Lines 8–11 of $\text{Delete}(\ell)$ that can be attributed to some specific label $\beta \in L(\mathcal{P})$ over the entire execution of $\text{BuildNT}(U_{\text{root}})$. Each execution of Lines 8–11 for β performs k_β operations on map fields. Let $w_r(\beta)$ be the weight of the connected component containing β at the beginning of the loop of Lines 8–11, at the r th time that β is considered in those lines; thus, $w_0(\beta) \leq \sum_{\ell \in L(\mathcal{P})} k_\ell$. Then, $w_r(\beta) \leq w_0(\beta)/2^r$. The reason is that we only consider β if (i) β is contained in one of the two components that result from deleting an edge in Line 3 and (ii) the component containing β has the smaller weight of the two components. Thus, the number of times β is considered in Lines 8–11 over the entire execution of $\text{BuildNT}(U_{\text{root}})$ is $O(\log w_0(\beta))$, which is $O(\log M_{\mathcal{P}})$, since $w_0(\beta) = O(M_{\mathcal{P}})$. Therefore, the total number of updates of map fields over all labels is $O(\log M_{\mathcal{P}} \cdot \sum_{\ell \in L(\mathcal{P})} k_\ell)$,

Algorithm 3: Delete(ℓ)

```
1 Let  $Y$  be the connected component of  $G_{\text{BNT}}$  that contains  $\ell$ 
2 foreach  $\alpha \in \text{Ch}(\ell)$  do
3   Delete edge  $\{\ell, \alpha\}$  from  $G_{\text{BNT}}$ 
4   if  $Y$  remains connected then continue Let  $Y_1, Y_2$  be the connected components of  $G_{\text{BNT}}$ ; assume
      that  $Y_1.\text{weight} \leq Y_2.\text{weight}$ 
5   Let  $Y_1.\text{map} = \text{null}$  and  $Y_1.\text{semiU} = \text{null}$ 
6   Let  $Y_2.\text{map} = Y.\text{map}$  and  $Y_2.\text{semiU} = Y.\text{semiU}$ 
7   Let  $J = \emptyset$ 
8   foreach  $\beta \in Y_1$  do
9     foreach  $i \in [k]$  such that  $\beta \in L(\mathcal{T}_i)$  do
10      Move  $\beta$  from  $Y_2.\text{map}(i)$  to  $Y_1.\text{map}(i)$ 
11       $J = J \cup \{i\}$ 
12   foreach  $i \in J$  do
13     foreach  $j \in \{1, 2\}$  do
14       if  $Y_j.\text{map}(i) = \emptyset$  then
15         Delete  $Y_j.\text{map}(i)$ 
16       else if  $Y_j.\text{map}(i)$  is a singleton then
17         Let  $\gamma$  be the single element in  $Y_j.\text{map}(i)$ 
18          $\gamma.\text{count} = \gamma.\text{count} + 1$ 
19         if  $\gamma.\text{count} = k_\gamma$  then add  $\gamma$  to  $Y_j.\text{semiU}$ 
20 Delete  $\ell$  from  $G_{\text{BNT}}$ 
```

which is $O(M_{\mathcal{P}} \log M_{\mathcal{P}})$. It can be verified that the number of updates to `count` and `semiU` fields is also $O(M_{\mathcal{P}} \log M_{\mathcal{P}})$. A similar analysis shows that the total time to update `weight` fields over all edge deletions performed by $\text{BuildNT}(U_{\text{root}})$ is $O(M_{\mathcal{P}} \log M_{\mathcal{P}})$.

To summarize, the work done by $\text{BuildNT}(U_{\text{root}})$ consists of three parts: (i) initialization, (ii) maintaining connected components, and (iii) maintaining the `weight`, `map`, and `semiU`, and fields for each connected component, as well as `count` for each label ℓ . Part (i) takes $O(M_{\mathcal{P}})$ time. Part (ii) involves $O(M_{\mathcal{P}})$ edge and node deletions on the HDT data structure, at an amortized cost of $O(\log^2 M_{\mathcal{P}})$ per deletion. Part (iii) requires a total of $O(M_{\mathcal{P}} \log M_{\mathcal{P}})$ updates to the various fields. Using data structures that take logarithmic time per update, leads to our main result.

Theorem 2. *BuildNT can be implemented so that $\text{BuildNT}(U_{\text{root}})$ runs in $O(M_{\mathcal{P}} \log^2 M_{\mathcal{P}})$ time.*

6 Discussion

Like our earlier algorithm for compatibility of ordinary phylogenetic trees, the more general algorithm presented here, BuildNT , is a polylogarithmic factor away from optimality (a trivial lower bound is $\Omega(M_{\mathcal{P}})$, the time to read the input). BuildNT has a linear-space implementation, using the results of Thorup [22]. A question to be investigated next is the performance of the algorithm on real data. Another important issue is integrating our algorithm into a synthesis method that deals with incompatible profiles.

References

- [1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.*, 10(3):405–421, 1981.
- [2] B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992.
- [3] V. Berry and C. Semple. Fast computation of supertrees for compatible phylogenies with nested taxa. *Systematic Biology*, 55(2):270–288, 2006.
- [4] O. R. P. Bininda-Emonds, M. Cardillo, K. E. Jones, R. D. E. MacPhee, R. M. D. Beck, R. Grenyer, S. A. Price, R. A. Vos, J. L. Gittleman, and A. Purvis. The delayed rise of present-day mammals. *Nature*, 446:507–512, 2007.
- [5] M. Bordewich, G. Evans, and C. Semple. Extending the limits of supertree methods. *Annals of Combinatorics*, 10:31–51, 2006.
- [6] D. Bryant and J. Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351:296–302, 2006.
- [7] P. Daniel and C. Semple. Supertree algorithms for nested taxa. In O. R. P. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, pages 151–171. Kluwer, Dordrecht, 2004.
- [8] Y. Deng and D. Fernández-Baca. Fast compatibility testing for rooted phylogenetic trees. In *Proceedings of 27th Annual Symposium on Combinatorial Pattern Matching*, to appear.
- [9] S. Even and Y. Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, Jan. 1981.
- [10] M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24:1–13, 1999.
- [11] C. E. Hinchliff, S. A. Smith, J. F. Allman, J. G. Burleigh, R. Chaudhary, L. M. Coghill, K. A. Crandall, J. Deng, B. T. Drew, R. Gazis, K. Gude, D. S. Hibbett, L. A. Katz, H. D. Laughinghouse IV, E. J. McTavish, P. E. Midford, C. L. Owen, R. H. Reed, J. A. Reesk, D. E. Soltis, T. Williams, and K. A. Cranston. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. *Proceedings of the National Academy of Sciences*, 112(41):12764–12769, 2015.
- [12] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001.
- [13] R. M. Page. Taxonomy, supertrees, and the tree of life. In O. R. P. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, pages 247–265. Kluwer, Dordrecht, 2004.
- [14] I. Pe’er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM J. Comput.*, 33(3):590–607, 2004.

- [15] M. A. Ragan. Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution*, 1:53–58, 1992.
- [16] M. J. Sanderson. Phylogenetic signal in the eukaryotic tree of life. *Science*, 321(5885):121–123, 2008.
- [17] E. W. Sayers et al. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 37(Database issue):D5–D15, 2009.
- [18] C. Semple and M. Steel. *Phylogenetics*. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford, 2003.
- [19] S. A. Smith, J. W. Brown, and C. E. Hinchliff. Analyzing and synthesizing phylogenies using tree alignment graphs. *PLoS Computational Biology*, 9(9):e1003223, 2013.
- [20] M. A. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *J. Classification*, 9:91–116, 1992.
- [21] The Angiosperm Phylogeny Group. An update of the Angiosperm Phylogeny Group classification for the orders and families of flowering plants: APG IV. *Botanical Journal of the Linnean Society*, 181:1–20, 2016.
- [22] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 343–350. ACM, 2000.